A Short Introduction to Monte Carlo Tree Search in RL

Omar Darwiche Domingues
SequeL team, Inria Lille - Nord Europe, France

Acknowledgment: based on Emilie Kaufmann's slides
http://chercheurs.lille.inria.fr/ekaufman/RLCours8.pdf

Outline

Motivation

Sparse Sampling

UCT - UCB applied to Trees

AlphaZero

Summary

Dynamic Programming and Reinforcement Learning

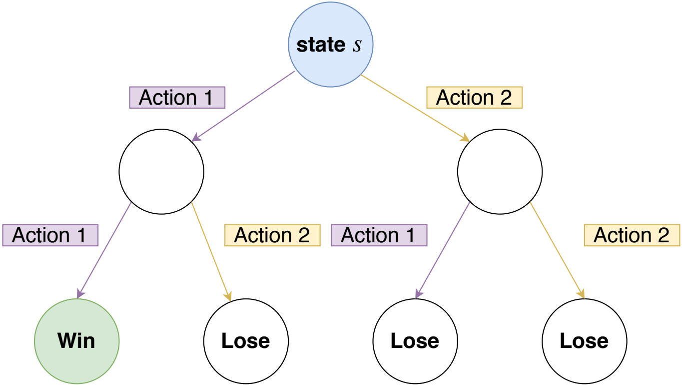
- Environment: Markov decision process $(\mathcal{S}, \mathcal{A}, P, r)$
- Agent: policy $\pi : \mathcal{S} \to \mathcal{A}$
- Some examples of algorithms according to the model available (full model or simulator) and to the size of the MDP:

Model	Small MDPs	Large MDPs
Full	Value/Policy Iteration	Approximate VI/PI
Simulator	Q-Learning, SARSA	FVI, DQN, policy gradient

- These algorithms compute either a value function $Q(s, a)$ or a policy $\pi(a|s)$ for all states s and actions a.

Tree Search Algorithms

- Take a fixed state s,
- Simulate possible outcomes starting from s,
- Choose an action that is expected to give the best outcome.



Tree Search Algorithms

We can define a policy using a tree search algorithm:

- At a state s_t, use tree search strategy to select a_t,
- Play a_t and observe s_{t+1} and r_t,
- $t \leftarrow t + 1$.

Outline

Motivation

Sparse Sampling

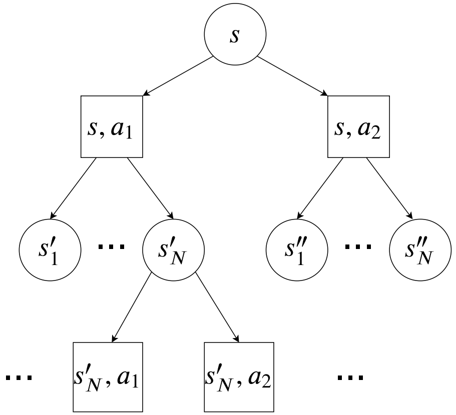
UCT - UCB applied to Trees

AlphaZero

Summary

Sparse Sampling

- Proposed by Kearns et al. (2002).
- Goal: estimate $Q(s, a)$ for all actions a in a fixed state s.
- For each state s and each action a, sample N transitions.



Sparse Sampling

- It is a recursive algorithm.
- From a node (s, a), sample N transitions $(R_i, S_i')_{i=1}^{N}$.
- At depth h in the tree, compute

$$\widehat{Q}_h(s, a) = \frac{1}{N} \sum_{i=1}^{N} \left(R_i + \gamma \widehat{V}_{h+1}(S_i') \right)$$
$$\widehat{V}_h(s) = \max_a \widehat{Q}_h(s, a)$$

- Initialization (for accuracy ϵ):

$$V_{H+1}(s) = 0, \quad \text{for all } s, \quad \text{where} \quad H = \left\lceil \frac{\log(\epsilon(1-\gamma))}{\log \gamma} \right\rceil$$

- Error $= \left| \widehat{Q}_1(s, a) - Q^*(s, a) \right| \approx \epsilon$ for a choice of N that depends on ϵ and γ.

Sparse Sampling

Problems:

- N can be very large: the algorithm can be very slow.
- If we stop the algorithm before it terminates, we cannot recommend an action.
- Action selection in the tree is not adaptive.

Outline

Motivation

Sparse Sampling

UCT - UCB applied to Trees

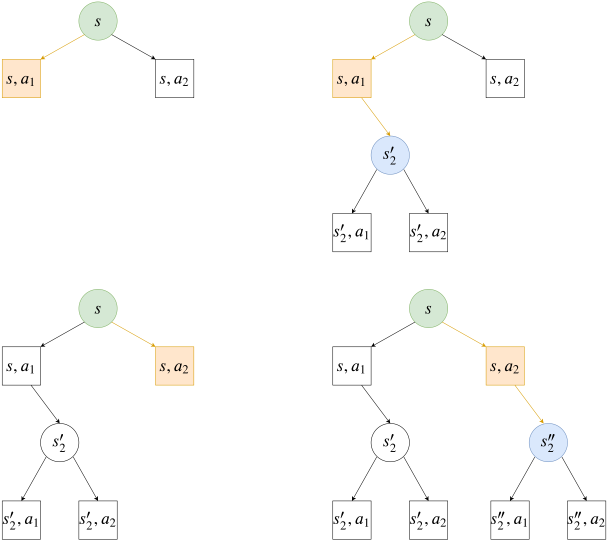
AlphaZero

Summary

The UCT Algorithm

- Proposed by Kocsis and Szepesvári (2006).
- Trajectory-based algorithm (not recursive).
- Adaptive selection of actions.
- For each node (s, a), we store:
  - $N(s, a)$: the number of times we visited the node.
  - $S(s, a)$: the total sum of rewards obtained after visiting the node.

The UCT Algorithm



The UCT Algorithm

- In a node s, we select the action:

$$a(s) \in \operatorname*{argmax}_a \underbrace{\frac{S(s,a)}{N(s,a)}}_{\text{estimated reward}} + \underbrace{c\sqrt{\frac{\log\left(\sum_b N(s,b)\right)}{N(s,a)}}}_{\text{exploration bonus}}$$

- After a number of iterations, we need to recommend an action. Possibilities:
  - Choose the action which has been played the most at the root.
  - Choose the action with largest estimated value.
  - Sample an action with a probability that depends on its number of pulls at the root.

The UCT Algorithm

Node initialization: how to estimate the value of a newly added node?

- Roll-out: Monte Carlo evaluation of a policy (e.g., random policy).
- Optimistic initialization: use maximum possible value.
- Heuristic: problem-specific evaluation function.

Outline

Motivation

Sparse Sampling

UCT - UCB applied to Trees

AlphaZero

Summary

AlphaZero

Limitations of UCT:

- Using UCT to define a policy can be to slow: we need to run it for every new state we observe.
- Designing a good way to evaluate new nodes can be difficult.

AlphaZero (Silver et al., 2017):

- Algorithm for playing games that uses MCTS.
- Network of parameters θ such that $(p, v) = f_\theta(s)$
  - p is a probability vector over possible actions in state s
  - v is the estimated value of the state s.
- Uses p to improve tree search and uses v as evaluation of leaf nodes.
- Train the network by using the MCTS policy (to improve p) and the observed outcomes (to improve v).

AlphaZero - MCTS

- Action selection in MCTS:

$$a(s) \in \operatorname*{argmax}_a \frac{S(s,a)}{N(s,a)} + c\, P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$

- Output of MCTS:

$$\pi(a) = \frac{N(s_{\mathrm{root}}, a)^{1/\tau}}{\sum_b N(s_{\mathrm{root}}, b)^{1/\tau}}$$

AlphaZero - Training the network

Let θ be the current parameters of the network and $(p, v) = f_\theta(s)$.

1. generate N games where each player uses MCTS(θ) to select the next action a_t (and output a probability over actions π_t)

$$\mathcal{D} = \bigcup_{i=1}^{\text{Nb games}} \left\{ \left(s_t, \pi_t, \pm r_{T_i}\right) \right\}_{i=1}^{T_i}$$

   T_i: length of game i, $r_{T_i} \in \{-1, 0, 1\}$ outcome of game i for one player

2. Based on a sub-sample of $\mathcal{D}$, train the neural network using stochastic gradient descent on the loss function

$$L(s, \pi, z; p, v) = (z - v)^2 - \pi^\top \ln(p) + c\|\theta\|^2$$

A nice actor-critic architecture

AlphaZero alternates between

- **The actor**: MCTS(θ)
  - Generates trajectories guided by the network f_θ but still exploring
  - can be seen as a policy improvement step
- **The critic**: neural network f_θ
  - updates θ based on trajectories followed by the actor
  - evaluates the actor's policy

Outline

Motivation

Sparse Sampling

UCT - UCB applied to Trees

AlphaZero

Summary

Summary

- MCTS: recommend an action to take in a fixed state s.
- When the state space is very large, MCTS can be a good alternative to algorithms that estimate a value function or a policy for all states.

Kearns, M., Mansour, Y., and Ng, A. (2002). A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes.

Kocsis, L. and Szepesvári, C. (2006). Bandit-based Monte-Carlo planning. In European Conference on Machine Learning.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815.